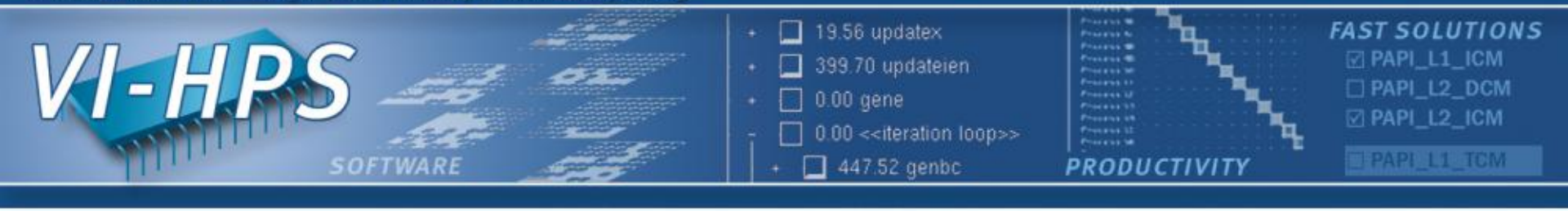

VAMPIRTRACE ABOUT MANUAL INSTRUMENTATION

DKRZ Tutorial 2012 in Hamburg
June, 2012

Ronny Tschüter

Slides by: Andreas Knüpfer, Jens Doleschal,

ZIH, Technische Universität Dresden

Source Code Instrumentation

- Instrumentation: Process of modifying programs to detect and report events

- There are various ways of instrumentation:
  - Manually
    - Large effort, error prone
    - Difficult to manage
  - Automatically
    - Via source to source translation
    - Via compiler instrumentation
    - Program Database Toolkit (PDT)
    - OpenMP Pragma And Region Instrumenter (Opari)

Source Code Instrumentation

```
int foo(void* arg) {

if (cond) {

        return 1;
}

return 0;
}
```

```
int foo(void* arg) {
enter(7);
if (cond) {
        leave(7);
        return 1;
}
leave(7);
return 0;
}
```

manually or automatically

Source Code Instrumentation

- Wrapper option `-vt:inst <insttype>` specifies the instrumentation type to be used
  - `compinst`
    Fully-automatic instrumentation by the compiler
  - `manual`
    Manual instrumentation by using VampirTrace's API
    (needs source-code modifications)
  - `tauinst`
    Fully-automatic instrumentation by the tau_instrumentator
  - `dyninst`
    Binary-instrumentation with Dyninst

Using the VampirTrace API

- The `VT_USER_START`, `VT_USER_END` calls can be used to instrument any user-defined sequence of statements

Fortran:

```
#include "vt_user.inc“


VT_USER_START(’name’)

...

VT_USER_END(’name’)
```

C:

```
#include "vt_user.h“


VT_USER_START("name");

...

VT_USER_END("name");
```

- If a block has several exit points, all exit points have to be instrumented with `VT_USER_END`

Using the VampirTrace API

- For C++: only entry points into a scope need to be marked

C++:

```
#include "vt_user.h“
{
VT_TRACER("name");
...
}
```

Using the VampirTrace API

- Instrumented sources have to be compiled with `-DVTRACE`
  - combined with automatic compiler instrumentation:

```
% vtcc -DVTRACE hello.c -o hello
```

  - without compiler instrumentation:

```
% vtcc -vt:inst manual -DVTRACE hello.c -o hello
```

On Blizzard use following flags for Fortran:

-WF,-DVTRACE -I$(VT_INC)

- Note:
  The option `-vt:inst manual` can be used with non-instrumented sources. Binaries created in this manner only contain MPI and OpenMP instrumentation, which might be desirable in some cases.

Measurement Controls

- Switching tracing on/off
  - Use the `VT_ON/ VT_OFF` instrumentation calls to start and stop the recording of events

```
int main()
{
        ...
        VT_OFF();
        initialize();
        VT_ON();
        compute();
        ...
}
```

  - To check whether if tracing is enabled or not use the call `VT_IS_ON`

- Trace buffer rewind
  - Useful when the program should decide dynamically after a specific code section (i.e. a time step or iteration) if this section has been interesting (i.e. anomalous/slow behavior) and should be recorded to the trace file
  - Use the instrumentation call `VT_SET_REWIND_MARK` at the beginning of a (possibly not interesting) code section
  - Later, you can decide to rewind the trace buffer to the mark with the call `VT_REWIND`

```
do step=1,number_of_time_steps
        VT_SET_REWIND_MARK()
        call compute_time_step(step)
        if(finished_as_expected) VT_REWIND()
end do
```

- Intermediate buffer flush
  - In addition to an automated buffer flush when the buffer is filled, it is possible to flush the buffer at any point of the application. This way you can guarantee that after a manual buffer flush there will be a sequence of the program with no automatic buffer flush interrupting. To flush the buffer you can use the call `VT_BUFFER_FLUSH`.

- **Note**:
  - Compile with `-DVTRACE`
  - If the sources contains further VampirTrace API calls and only the calls for measurement controls shall be disabled, then the sources have to be compiled with `-DVTRACE_NO_CONTROL`.